游荡在思考的迷宫中

甄景贤 (King-Yin Yan) and Juan Carlos Kuri Pinto

General.Intelligence@Gmail.com

Abstract. 介绍 Itamar Arel 在 2012 年提出的一个基於增强学习和深度学习的 cognitive architecture [1]，及其与控制论、Hamiltonian 系统的关系。

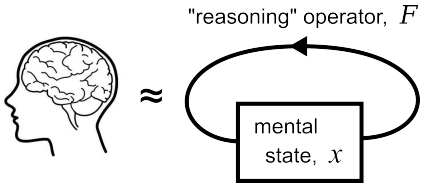
本文并没有原创内容，只是以 tutorial 的形式介绍一些已知的理论，及提供一个新的观点，或许对 AGI 的发展有帮助。

1 中心思想

标题中的**比喻**是指用增强学习的方法控制一隻自主的智能系统 (autonomous agent)，在「思维空间」中寻找最优路径：

(1)

关键是将「思考」看成是一个动态系统 (dynamical system)，它运行在思维状态 (mental states) 的空间中：



(2)

举例来说，一个思维状态可以是以下的一束命题：

– 我在我的房间内，正在写一篇 AGI-16 的论文。

– 我正在写一句句子的开头：「举例来说，....」
– 我将会写一个 NP (noun phrase)：「一个思维状态....」

思考的过程就是从一个思维状态 过渡 (transition) 到另一个思维状态。就算我现在说话，我的脑子也是靠思维状态记住我说话说到句子结构的哪部分，所以我才能组织句子的语法。

思维状态是一支向量 $\boldsymbol{x} \in X$，X 是所有可能的思维状态，思考算子 (reasoning operator) R 是一个 endomorphism 映射：$X \to X$。

换句话说：我们将逻辑 AI 的整套器材搬到向量空间中去处理。这个做法，部分是受到 Google 的 PageRank [4] 和 Word2Vec [3] 算法的启发，因为它们都是在向量空间中运作，而且非常成功。

2 经典逻辑 AI

Strong AI 的问题在理论上已经被数理逻辑完整地描述了，馀下的问题是学习算法，因为在逻辑 AI 的架构下，学习算法很慢（复杂性很高），这就是我们要解决的。

我研究 logic-based AI 很多年，因此我的思路喜欢将新问题还原到逻辑 AI 那边去理解，但实际上我提倡的解决办法不是靠经典逻辑，甚至不是 symbolic 的。但在这篇文章我还是会经常跳回到逻辑 AI 去方便理解。

用数理逻辑模拟人的思想是可行的，例如有 deduction, abduction, induction 等这些模式，详细可见《Computational logic and human thinking》by Robert Kowalski, 2011. 这些方面不影响本文的阅读。值得一提的是，作者 Kowalski 是 logic programming，特别是 Prolog，的理论奠基人之一。

在经典逻辑 AI 中，「思考」是透过一些类似以下的步骤：

$$\text{前提} \vdash \text{结论} \tag{3}$$

$$\boxed{\text{今天早上下雨}} \vdash \boxed{\text{草地是湿的}} \tag{4}$$

亦即由一些命题(propositions) 推导到另一些命题。

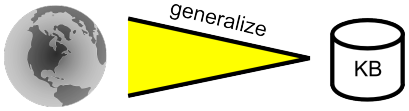
推导必须依靠一些逻辑的法则命题 (rule propositions)，所谓「法则」是指命题里面带有 x 这样的变量(variables)：

$$\boxed{\text{地方 x 下雨}} \wedge \boxed{\text{x 是露天的}} \vdash \boxed{\text{地方 x 是湿的}} \tag{5}$$

这些法则好比「逻辑引擎」的燃料，没有燃料引擎是不能推动的。

注意：命题里面的 x，好比是有「洞」的命题，它可以透过 substitution 代入一些实物 (objects)，而变成完整的命题。这种「句子内部」(sub-propositional) 的结构可以用 predicate logic （谓词逻辑）表达，但暂时不需要理会这些细节。

Logic-based AI 可以看成是将世界的「模型」压缩成一个「知识库」(knowledge-base, KB)，里面装著大量逻辑式子：



(6)

世界模型是由大量的逻辑式子经过组合而生成的，有点像向量空间是由其「基底」生成；但这生成过程在逻辑中特别复杂，所以符号逻辑具有很高的压缩比，但要学习一套逻辑知识库，则相应地也有极高的复杂度。

3 控制论

以下内容可以在一般「现代控制论」教科书中找到，例如：

- Daniel Liberzon 2012: *Calculus of variations and optimal control theory – a concise introduction*
- 李国勇 2008：《最优控制理论与应用》
- 张洪钺、王青 2005：《最优控制理论与应用》

一个**动态系统 (dynamical system)** 可以用以下方法定义：

$$\text{离散时间：} \quad \boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t) \tag{7}$$

$$\text{连续时间：} \quad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{8}$$

其中 f 也可以随时间改变。如果 f 不依赖时间，则系统是 time-invariant （定常的），形式上如 (8) 那种微分方程叫作 autonomous （自主的）。

在我的智能系统理论里，我把 F 或 f 设定成 RNN (recurrent neural network)，即反馈式神经网络：

$$\text{离散时间：} \quad \boldsymbol{x}_{t+1} = \boxed{\text{RNN}}(\boldsymbol{x}_t) \tag{9}$$

$$\text{连续时间：} \quad \dot{\boldsymbol{x}} = \boxed{\text{RNN}}(\boldsymbol{x}) \tag{10}$$

这里 recurrent 指的是它不断重复作用在 $\boldsymbol{x}$ 之上，但实际上它是一个普通的前馈式 (feed-forward) 神经网络。注意：在抽象理论中，f 和 F 可以是任意函数，我把它们设计成 NN 只是众多可能的想法之一。之所以选用 NN，是因为它有 universal function approximator 的功能，而且是我们所知的最「聪明」的学习机器之一。

在我提出的智能系统里，$\dot{x}$ 是由學習機器給出的，换句話說，$\dot{x}$ 是思維狀態在梯度下降至最佳狀態時的方向導數。

一个（连续时间的）控制系统 (control system) 定义为：

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{11}$$

其中 $\boldsymbol{u}(t)$ 是控制向量。控制论的目的就是找出最好的 $\boldsymbol{u}(t)$ 函数，令系统由初始状态 $\boldsymbol{x}_0$ 去到终点状态 $\boldsymbol{x}_\perp$。

注意：人工智能中的 A* search，是动态规划的一个特例。换句话说，用动态规划在某个空间中「漫游」，可以模拟到 best-first 搜寻的功能。

在这框架下，智能系统的运作可以分开成两方面：思考 和 学习。

思考即是根据已学得的知识（知识储存在 RNN 里），在思维空间中找寻 $\boldsymbol{x}$ 最优的轨迹，方法是用控制论计算 $\boldsymbol{u}^*$。$\boldsymbol{x}$ 的轨迹受 RNN 约束（系统只能依据「正确」的知识去思考），但思考时 RNN 是不变的。

学习就是学习神经网络 RNN 的 weights W。此时令 $u = 0$，即忽略控制论方面。

以上两者是两个独立的方面，但不排除它们可以在实际中同时进行。

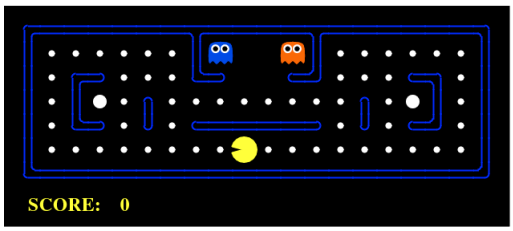
4 什么是强化学习？

Reinforcement learning 是机器学习里面的一个分支，特别善於控制一只能够在某个环境下 自主行动 的个体 (autonomous agent)，透过和 环境 之间的互动，例如 sensory perception 和 rewards，而不断改进它的 行为。

听到强化学习，你脑里应该浮现一只甲由那样的小昆虫，那就是 autonomous agent 的形象：

(12)

对「环境」(environment) 这概念，你应该想到像以下这经典游戏的迷宫：



(13)

包括有追捕你的怪物、和吃了会加分的食物（这些代表负值和正值的 rewards）。当然，实际应用的「环境」和「奖励」可以是抽象的，这游戏是一个很具体的例子。

4.1 输入 / 输出

记住，reinforcement learning 的 输入 是：

- 状态 (States) = 环境，例如迷宫的每一格是一个 state
- 动作 (Actions) = 在每个状态下，有什么行动是容许的
- 奖励 (Rewards) = 进入每个状态时，能带来正面或负面的价值 (utility)

而输出就是：

- 方案 (Policy) = 在每个状态下，你会选择哪个行动？

於是这 4 个元素的 tuple $(S\ A\ R\ P)$ 就构成了一个强化学习的系统。在抽象代数中我们常常用这 tuple 的方法去定义系统或结构。

再详细一点的例子就是：

- states S = 迷宫中每一格的位置，可以用一对座标表示，例如 (1,3)
- actions A = 在迷宫中每一格，你可以行走的方向，例如：{ 上，下，左，右 }
- rewards R = 当前的状态 (current state) 之下，迷宫中的那格可能有食物 (+1) 、也可能有怪兽 (-100)
- policy P = 一个由状态 $\rightarrow$ 行动的函数，意即：这函数对给定的每一个状态，都会给出一个行动。

(S, A, R) 是使用者设定的，P 是算法自动计算出来的。

4.2 人与虫之间

第一个想到的问题是：为什么不用这个方法打造人工智能？但现时的强化学习算法，只对比较细小和简单的环境适用，对於大的复杂的世界，例如象棋的 10^{xxx} 状态空间，仍是 intractable 的。

关键就是，高等智慧生物会在脑中建立世界的模型 (world model) 或知识 (knowledge)，而强化学习只是关心简单的「状态－行动」配对。

强化学习的领导研究者 Richard Sutton 认为，只有这种学习法才考虑到自主个体、环境、奖励等因素，所以它是人工智能中最 top-level 的 architecture，而其他人工智能的子系统，例如 logic 或 pattern recognition，都应该在它的控制之下，我觉得颇合理。

(14)

所以要制造 strong AI，一个可能的方案就是结合强化学习和某种处理复杂 world model 的能力：

(15)

「你们已经由虫进化成人，但在你们之内大部份仍是虫。」

— 尼采, Thus spoke Zarathustra

「如果人类不相信他们有一天会变成神，他们就肯定会变成虫。」

— Henry Miller

4.3 強化學習的原理

《AI — a modern approach》这本书第 21 章有很好的简介。《AIMA》自然是经典，很多人说他们是读这本书而爱上 AI 的。这本书好处是，用文字很耐性地解释所有概念和原理，思路很清晰，使读者不致有杂乱无章的感觉。例如 21 章首先讲 passive reinforcement learning，意思是当 policy 是固定时，纯粹计算一下 agent 期望的价值（utility，即 rewards 的总和）会是多少。有了这基础后再比较不同 policies 的好坏。这种思路在数学中很常见：首先考虑简单到连白痴也可以解决的 case，然后逐步引入更多的复杂性。例如数学归纳法，由 $N = 1$ 的 case 推到 $N \to \infty$。

为免重复，我只解释到明白 Q learning 的最少知识。

4.4 Utility (价值，或效)

U 是一连串行动的 rewards 的总和。例如说，行一步棋的效用，不单是那步棋当前的利益，还包括走那步棋之后带来的后果。例如，当下贪吃一只卒，但 10 步后可能被将死。又或者，眼前有美味的食物，但有些人选择不吃，因为怕吃了会变肥。

一个 state 的效用 U 就是：假设方案固定，考虑到未来所有可能的 transitions，从这个 state 开始的平均期望的 total reward 是多少:

$$U(S_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \ R(S_t)\ ]$$

其中 $\mathbb{E}[\,]$ 代表期望值，γ 是 discount factor，例如 0.9 或什么。

实例：考虑这简单的迷宫：

那些箭咀表示的是众多可能方案中的其中一个。

根据这个方案，由 $(1,1)$ 开始的运行可能是这个结果：

下面橙色的值是每个 state 的 reward。在这例子中，每个不是终点的格，也会扣 0.04 分。

但从同一起点，同一方案，也可以出现不同结果，例如在 (1,3) 企图向右爬，但实际结果是向下跌一格；这些 state transitions 是由外在世界的机率决定的。（例如某人读了大学文凭，但遇上经济不景，他的薪水未必能达到行动的预期效果。）

同一方案的运行结果可以是：

或者：

4.5 Bellman condition

这是 dynamic programming（动态规划）的中心思想，又叫 Bellman optimality condition。

在人工智能里我们叫 reinforcement learning，但在控制论的术语里叫 dynamic programming，两者其实是一样的。Richard Bellman 在 1953 年提出这个方程，当时他在 RAND 公司工作，处理的是运筹学的问题。他也首先使用了"curse of dimensionality" 这个术语，形容动态规划的主要障碍。

考虑的问题是：要做一连串的 sequential decisions。

Bellman equation 说的是：「如果从最佳选择的路径的末端截除一小部分，馀下的路径仍然是最佳路径。」

换句话说，如果一系列的选择 A B C D E.... 是最优的，那么这系列除去开始的 A，那 B C D E.... 系列应用在后继的状态上也是最优的。

（例如，你从香港乘车到北京，选择了最便宜的路线，此路线经过 10 个车站，第二站是深圳：

$$\text{香港} \to \text{深圳} \to \ldots\ldots \to \text{北京}$$

但如果除去出发点香港站，那么由第二站深圳到最后的北京站：

$$\text{深圳} \to \ldots\ldots \to \text{北京}$$

这路线仍然是馀下 9 个站之间最便宜的。）

用数学表示：

$$U^*(S) = \max_a \{R(a) + U^*(S')\}$$

$$U^*(\text{全路径}) = \max_a \{R(\text{在当前状态下选取 a}) + U^*()\}$$

$*$ 表示 最优 (optimal)。这条看似简单的式子是动态规划的全部内容；它的意义是：我们想获得最佳效益的路径，所以将路径切短一些，於是问题化解成一个较小的问题；换句话说它是一个 recursive relation。

4.6 Delta rule

这只是一个简单的 trick，在机器学习中经常出现。假设我们有一个理想，我们要逐步调教当前的状态，令它慢慢趋近这个理想。方法是：

$$\text{当前状态} := \text{当前状态} + \alpha(\text{理想} - \text{当前状态})$$

其中 α 叫「学习速度 (learning rate)」。"Delta" (Δ) 指的是理想和现状之间的差异。

很明显，只要反覆执行上式，状态就会逐渐逼近理想值。

（Delta rule 的微分形式就是我们熟悉的「梯度下降」：$x \mathrel{+}= \eta \cdot \frac{dy}{dx}$）

4.7 Temporal difference (TD) learning

将 delta rule 应用到 Bellman condition 上，去寻找最优路径，这就是 temporal difference learning。

我们还是从简单情况开始：假设方案固定，目标是学习每个 state 的 utility。

理想的 $U(S)$ 值，是要从 state S 开始，试验所有可能的 transitions，再计算这些路径的 total rewards 的平均值。

但实际上，agent 只能够每次体验一个行动之后的 state transition。

所以要应用 Bellman condition：一个 state S 的 U 值，是它自身的 reward，加上所有可能的后继 states 的 U 值，取其机率平均，再乘以 discount factor γ：

$$U(S) = R(S) + \gamma \sum_{S'} P(S \to S')\, U(S')$$

其中 P 是 transition 的机率，S' 是后继 state，$\sum$ 是对所有后继 states 求和。换句话说，这是理想的 $U(S)$ 和 $U(S$的后继$)$ 之间的关系，是一个 recursive relation。

例如，假设 agent 现时对 state (1,3) 和 state (2,3) 的估值，分别为 0.84 和 0.92。又假设 agent 察觉到，根据现有方案，在 (1,3) 时总是会发生跳到 (2,3) 这个 transition。那么这两个 states 的 U 值，应该符合这条约束：

$$U(1,3) = -0.04 + U(2,3)$$

换句话说，这是两个 states 之间，U 值的 local (局部的) 约束。

TD learning 的思想是：假设其他 $U(S')$ 的值正确，利用 Bellman optimality 来调整当下 state 的 $U(S)$。当尝试的次数多了，所有 U 值都会趋向理想。Agent 只需要用这条 update rule：

$$U(S) \mathrel{+}= \alpha(\ R(S) + \gamma U(S') - U(S)\ )$$

α 是 learning rate，它决定学习的速度（但它不能太大，避免 overshooting）。后面那东西是 $U(S)$ 和 $U(S)$ 的估值 (estimation) 之间的差别。对於理想的 $U(S)$ 和 $U(S')$，那差别会是 0。而在每个 time step，我们只是用 α 部分地调整这个差别。

最后一提，在上面理想约束的公式里，有对於机率 P 的求和，但在 update formula 中 P 不见了。那是因为 agent 在环境中的行动，暗含了对於 state transition 机率的 sampling (随机地取样本)。换句话说，那机率求和是由 agent 本身体现的。

P 是 state transitions 的机率，换句话是关於世界的一个 model。TD learning 不需要学习 P，所以叫 model-free learning。但正如开篇时说过，model-free 并不一定是好事，人的智慧就是基於我们对外在世界有一些很好的 models。

4.8 Q value

Q 值只是 U 值的一个变种; U 是对每个 state 而言的，Q 把 U 值分拆成每个 state 中的每个 action 的份量。换句话说，Q 就是在 state S 做 action A 的 utility。

Q 和 U 之间的关系是：

$$U(S) = \max_A Q(A, S)$$

Q 的好处是什么？下面将会介绍 active learning，而 Q value 配合 TD learning，可以在 active learning 中也消除 P，达到 model-free 的效果。

上面的 update rule 只要用这个关系改写就行：

$$U(S) \mathrel{+}= \alpha(\ R(S) + \gamma \max_{A'} Q(A', S') - Q(A, S)\ )$$

4.9 Active learning

在 passive learning 中，方案不变，我们已经能够计算每个 state S 的效用 $U(S)$，或者每个 state S 之下行动 A 的效用 $Q(S, A)$。

如果方案是可以改变的，我们只需计算不同方案的 Q 值，然后在每个 state S 选取相应於最大 Q 值的行动 A，那就是最佳方案，不是吗？

实际上执行的结果，却发现这些 agent 的方案很差！原因是，学习过程中的 Q 值是 estimate，不是理想的 Q 值，而如果根据这样的 Q 行动，agent 变得很短视，不会找到 optimal policy。（例如，某人经常吃同一间餐馆，但循另一路径走，可以发现更好的餐馆。）

Agent 需要尝试一些未知的状态 / 行动，才会学到 optimal policy；这就是所谓的 exploration vs exploitation （好奇心 vs 短暂贪婪）之间的平衡。

方法是，人工地将未知状态的价值增加一点：

$$U(S) = R(S) + \gamma \max_A \mathcal{F}[\ \sum_{S'} P(S \to S')U(S'),\ N(A, S)\ ]$$

其中 $N(A, S)$ 是状态 S 和行动 A 这对组合出现过（被经历过）的次数，$\mathcal{F}$ 是 exploration 函数，它平时回覆正常的 U 的估计值，但当 N 很小时（亦即我们对 S，A 的经验少），它会回覆一个比较大的估值，那代表「好奇心」的效用。

4.10 控制论与强化学习的关系

在强化学习中，我们关注两个数量：

- $R(\boldsymbol{x}, a)$ = 在状态 $\boldsymbol{x}$ 做动作 a 所获得的奖励(reward)
- $U(\boldsymbol{x})$ = 状态 $\boldsymbol{x}$ 的效用(utility) 或 价值 (value)

简单来说，「价值」就是每个瞬时「奖励」对时间的积分：

$$\boxed{\text{价值 U}} = \int \boxed{\text{奖励 R}}\, dt \tag{16}$$

（价值有时用 V 表示，但为避免和势能 V 混淆故不用。）

用控制论的术语，通常定义 cost functional：

$$J = \int L dt + \Phi(\boldsymbol{x}_\perp) \tag{17}$$

其中 L 是 "running cost"，即行走每一步的「价钱」；Φ 是 terminal cost，即到达终点 $\boldsymbol{x}_\perp$ 时，那位置的价值。

在分析力学里 L 又叫 Lagrangian，而 L 对时间的积分叫「作用量」：

$$\boxed{\text{作用量 (Action)}}\ S = \int L dt \tag{18}$$

Hamilton 的最小作用量原理 (principle of least action) 说，在自然界的运动轨迹里，S 的值总是取稳定值 (stationary value)，即比起邻近的轨迹它的 S 值最小。

所以有这些对应：

强化学习	最优控制	分析力学
效用/价值 U	价钱 J	作用量 S
即时奖励 R	running cost	Lagrangian L
action a	control u	（外力?）

用比较浅显的例子：和美女做爱能带来即时的快感 (= 奖励 R)，但如果强奸的话会坐牢，之后很长时间很苦闷，所以这个做法的长远价值 U 比其他做法较低，正常人不会选择它。

有趣的是，奖励 R 对应於力学上的 Lagrangian，其物理学单位是「能量」；换句话说，「快感」或「开心」似乎可以用「能量」的单位来量度，这和通俗心理学里常说的「正能量」不谋而合。而，长远的价值，是以 [能量 × 时间] 的单位来量度。

一个智能系统，它有「智慧」的条件，就是每时每刻都不断追求「开心能量」或奖励 R 的最大值，但它必需权衡轻重，有计划地找到长远的效用 U 的最大值。

4.11 经典分析力学 (analytical mechanics)

分析力学的物理内容，完全是牛顿力学的 $F = ma$，但在表述上引入了能量和 Hamiltonian 等概念，再使用微积分和变分法。

Lagrange 方程

Lagrange 引入了 Lagrangian $L = T - V$，可以分拆成動能 T 和勢能 V 兩部分。

重點是：動能 T 是速度 $\dot{\boldsymbol{x}}$ 的函數，勢能 V 是位置 $\boldsymbol{x}$ 的函數。

问题：如果在强化学习中的「快感 / 奖励」对应於 Lagrangian L，如何在奖励之中分拆出「动能」和「势能」的分量？

$$\boxed{\text{Lagrange equation}} \quad \frac{d}{dt}\frac{\partial L}{\partial \dot{x}_i} - \frac{\partial L}{\partial x_i} = 0 \tag{19}$$

这些方程的座标是 $(\boldsymbol{x}, \dot{\boldsymbol{x}})$，可以了解成位置空间 (configuration space) 上的 tangent bundle （下述）。

Hamilton 方程

Hamiltonian $H = T + V$，亦即总能量，但它表示成位置 $\boldsymbol{x}$ 和动量 $\boldsymbol{p}$ 的函数。

$$\boxed{\text{Hamilton equation}} \quad \begin{cases} \dot{\boldsymbol{x}} &= \dfrac{\partial H}{\partial p} \\ \dot{\boldsymbol{p}} &= -\dfrac{\partial H}{\partial x} \end{cases} \tag{20}$$

这些方程的座标是相位空间 (phase space) $(\boldsymbol{x}, \boldsymbol{p})$。

位置空间和相位空间之间的变换是 Legendre transformation：

$$\boxed{\text{tangent bundle}} \; TX \to T^*X \; \boxed{\text{cotangent bundle}} \tag{21}$$

$$(\boldsymbol{x}, \dot{\boldsymbol{x}}) \mapsto (\boldsymbol{x}, \boldsymbol{p}) \tag{22}$$

$$\boldsymbol{p} = \frac{\partial L}{\partial \dot{\boldsymbol{q}}} \quad \Rightarrow \quad H := \boldsymbol{p}\dot{\boldsymbol{x}} - L \tag{23}$$

Hamilton-Jacobi 方程

$$\boxed{\text{Hamilton-Jacobi equation}} \quad H(q, \frac{\partial S}{\partial q}, t) + \frac{\partial S}{\partial t} = 0 \tag{24}$$

其中 S 是「作用量」。下面我们会用动态规划的原理推导出此一方程。

Poisson 括号

$$\boxed{\text{Poisson 括号}} \quad \{F, H\} := \sum_i \{\frac{\partial F}{\partial q_i}\frac{\partial H}{\partial p_i} - \frac{\partial F}{\partial p_i}\frac{\partial H}{\partial q_i}\} \tag{25}$$

在力学系统中，它表示任意一力学量（函数 f）对时间的改变量：

$$\dot{f} = \{f, H\} \tag{26}$$

$$\frac{\partial}{\partial t} = \frac{\partial}{\partial p}\dot{p} + \frac{\partial}{\partial q}\dot{q} \tag{27}$$

以上使用了微分的 chain rule，但由於 $\dot{\boldsymbol{p}}$ 和 $\dot{\boldsymbol{q}}$ 可以由 Hamilton 方程 (20) 给出，所以得到 Poisson 括号的形式 (25)。

每个物理学生都知道的「经典－量子对应原理」：

$$[\boldsymbol{F}, \boldsymbol{G}] \quad \Leftrightarrow \quad i\hbar\{F, G\} \tag{28}$$

这对应原理是 P.A.M. Dirac 发现的。

在经典力学里，

$$\{\boldsymbol{x}, \boldsymbol{p}\} = 1 \tag{29}$$

但在量子力学里，

$$[\boldsymbol{X}, \boldsymbol{P}] = i\hbar \tag{30}$$

这也是 Heisenberg 测不准原理的由来：

$$\Delta\boldsymbol{X}\Delta\boldsymbol{P} \geq \frac{\hbar}{2} \tag{31}$$

如果在某一流形上，「广义」Poisson 括号是 nondegenerate （非退化）的，则它变成了辛流形结构的 $\omega = \{\cdot, \cdot\}$ 括号（下述）。

4.12 Hamiltonian 的出现

考虑一个典型的控制论问题，系统是：

$$\text{状态方程：} \quad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \tag{32}$$

$$\text{边值条件：} \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0\,,\ \boldsymbol{x}(t_\perp) = \boldsymbol{x}_\perp \tag{33}$$

$$\text{目標函数：} \quad J = \int_{t_0}^{t_\perp} L[\boldsymbol{x}(t), \boldsymbol{u}(t), t]dt \tag{34}$$

要找的是最优控制 $\boldsymbol{u}^*(t)$。

Lagrange multiplier 是找极大/小值的常用方法：如果我们要找：

$$\max\ f(x) \quad \text{subject to} \quad g(x) = 0 \tag{35}$$

Lagrange 建议我们建构 Lagrangian 函数：

$$L(x, \lambda) = f(x) - \lambda g(x) \tag{36}$$

然后求解：

$$\nabla_{x,\lambda} L = 0 \tag{37}$$

现在将 Lagrange multiplier 方法应用到我们的问题上，会发现新的目标函数是：

$$J = \int_{t_0}^{t_\perp} \{L + \boldsymbol{\lambda}^T(t)\,[f(\boldsymbol{x}, \boldsymbol{u}, t) - \dot{\boldsymbol{x}}]\}dt \tag{38}$$

因此可以引入一个新的标量函数 H，即 Hamiltonian：

$$H(\boldsymbol{x}, \boldsymbol{u}, t) = L(\boldsymbol{x}, \boldsymbol{u}, t) + \boldsymbol{\lambda}^T(t) f(\boldsymbol{x}, \boldsymbol{u}, t) \tag{39}$$

物理学上，$\boldsymbol{f}$ 的单位是速度，而 L 的单位是能量，所以 $\boldsymbol{\lambda}$ 应该具有 动量 的单位。

极小值原理

Lev Pontryagin (1908-1988) 提出了 极小值原理，是经典变分法的推广。经典变分法的最优条件是：

$$\frac{\partial H}{\partial \boldsymbol{u}} = \boldsymbol{0} \tag{40}$$

极小值原理将最优条件改成是：

$$\min_{u \in \Omega} H(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{u}, t) = H(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{u}^*, t) \tag{41}$$

即是说：在最优轨迹 $\boldsymbol{x}^*(t)$ 和最优控制 $\boldsymbol{u}^*(t)$ 上，H 取最小值。它的好处是，当 $\dfrac{\partial H}{\partial \boldsymbol{u}}$ 不连续或不存在时，或者 $\boldsymbol{u}$ 受其他约束时，也可以应用。

粗略来说，极小值原理比经典变分法更一般，而动态学习又比极小值原理更一般。

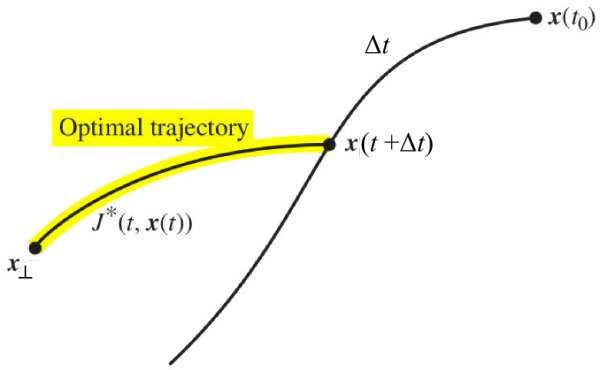
Hamilton-Jacobi-Bellman 方程

- Stanislaw Zak (2003): *Systems and control*

用动态规划的 **Bellman optimality condition** 可以推导出微分形式的 Hamilton-Jacobi-Bellman 方程。重温一下，Bellman 最优条件说的是：「从最优路径末端切去一小截之后，馀下的还是最优路径。」它通常写成如下的 recursive 形式：

$$\boxed{\text{最优路径}} = \text{在小段上选取最大奖励} + \boxed{\text{馀下的最优路径}} \tag{42}$$

$$J_t^* = \max_u \{ \boxed{\text{奖励(u, t)}} + J_{t-1}^* \} \tag{43}$$



(44)

我们在时间 interval 的开端切出一小段：

$$[t_0, t_\perp] = [t_0, t_0 + \Delta t] \cup [t_0 + \Delta t, t_\perp] \tag{45}$$

我们想优化的目标函数是：

$$J^*(t, \boldsymbol{x}, \boldsymbol{u}) = \min_u \{ \int_{t_0}^{t+\Delta t} L d\tau + \int_{t+\Delta t}^{t_\perp} L d\tau + \Phi(t_\perp, \boldsymbol{x}_\perp) \} \tag{46}$$

根据 Bellman 条件，目标函数变成：

$$J^*(t,\boldsymbol{x}) = \min_u\{\int_{t_0}^{t+\Delta t} L d\tau + J^*(t+\Delta t, \boldsymbol{x}(t+\Delta t))\} \tag{47}$$

用 Taylor series 展开右面的 J^*：

$$J^* + \frac{\partial J^*}{\partial t}\Delta t + \frac{\partial J^*}{\partial \boldsymbol{x}}(\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t)) + \text{H.O.T.} \tag{48}$$

左右两边的 J^* 互相消去，而且 $\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t) \approx \dot{\boldsymbol{x}}\Delta t$，於是有：

$$0 = \min_u\{\int_{t_0}^{t+\Delta t} L d\tau + \frac{\partial J^*}{\partial t}\Delta t + \frac{\partial J^*}{\partial \boldsymbol{x}}\dot{\boldsymbol{x}}\Delta t + \text{H.O.T.}\} \tag{49}$$

又由於 Δt 很小，而且 $\dot{\boldsymbol{x}} = \boldsymbol{f}$，所以：

$$0 = \min_u\{L\Delta t + \frac{\partial J^*}{\partial t}\Delta t + \frac{\partial J^*}{\partial \boldsymbol{x}}\boldsymbol{f}\Delta t + \text{H.O.T.}\} \tag{50}$$

全式除以 Δt 并令 $\Delta t \to 0$：

$$0 = \frac{\partial J^*}{\partial t} + \min_u\{L + \frac{\partial J^*}{\partial \boldsymbol{x}}\boldsymbol{f}\} \tag{51}$$

记得 Hamiltonian 的定义是 $H = L + \frac{\partial J^*}{\partial \boldsymbol{x}}\boldsymbol{f}$，所以得到想要的结果：

$$\boxed{\text{Hamilton-Jacobi equation}} \quad 0 = \frac{\partial J^*}{\partial t} + \min_u H \tag{52}$$

这个方程和量子力学中的 **Schrödinger equation** 很相似：

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t) = \left[V(x,t) + \frac{-\hbar^2}{2\mu}\nabla^2\right]\Psi(x,t). \tag{53}$$

其中 Ψ 类似於我们的 J （或许 Ψ 是自然界希望取极值的某种东西?）

4.13 Symplectic 结构

- Stephanie Singer (2001): *Symmetry in mechanics – a gentle, modern introduction*
- 锺万勰 2011：《力、功、能量与辛数学》

Symplectic 的拉丁文意思是「互相交错 (intertwined)」，它用来描述 Hamiltonian 系统的几何结构。中文译作「辛」是音译。Symplectic 概念是 Hermann Weyl 研究 Hamilton 系统的对称性时在 1939 年提出的。

在数值计算上，处理 Hamilton 系统时，如果算法尊重 symplectic 结构（叫 symplectic integrators），会比一般的算法更准确；而一般解微分方程的算法，例如 Euler 算法和 Runge-Kutta 算法，有时会给出错误的结果。

举例来说，从 Hamiltonian 的角度来看，动量 p (momentum) 和速度 v (velocity) 是成对偶的，p 总是伴随 v 出现，因为 $p \cdot v = mv^2$ 的单位是能量。

举另一个例子，假设我们有两个用来定义系统状态的向量：

$$x_1 = \begin{pmatrix} s_1 \\ f_1 \end{pmatrix}, \quad x_2 = \begin{pmatrix} s_2 \\ f_2 \end{pmatrix} \tag{54}$$

其中 s 是位移（单位是长度），f 是力。这两个向量的「辛内积」定义为：

$$\begin{aligned} \langle x_1, x_2 \rangle &= x_1^T \, J \, x_2 \\ &= \begin{pmatrix} s_1 \\ f_1 \end{pmatrix}^T \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix} \begin{pmatrix} s_2 \\ f_2 \end{pmatrix} \\ &= f_2 s_1 - f_1 s_2 \end{aligned} \tag{55}$$

其中矩阵 J 就是辛的微分形式 ω 的结构矩阵（下述）。由於 $f \cdot s$ 表示的是「所做的功」，上式表示的是

(状态 1 的力对状态 2 的位移所做的功)−
(状态 2 的力对状态 1 的位移所做的功) (56)

也就是「相互功」，辛正交则 $\langle x_1, x_2 \rangle = 0$，代表 work reciprocity（功的互等），所以辛几何是一种关於能量的代数。

在微分几何里，研究抽象的 Hamiltonian systems，会发现 symplectic 结构。这结构用微分流形 M 及其上的一个微分形式 (differential form) ω 来定义。需要一些微分几何的基础.....

Vectors and co-vectors

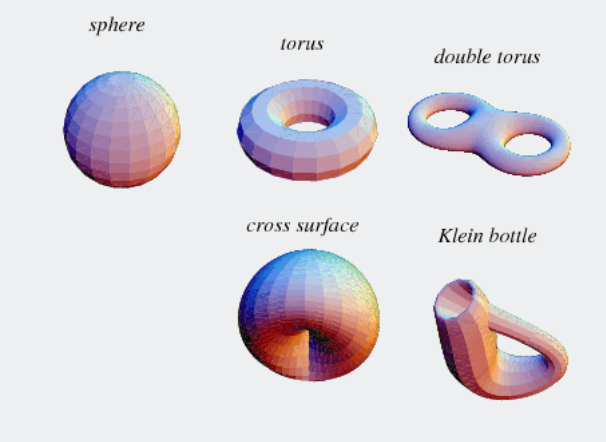
Vector 和 co-vector 之间的关系，可以看成是「d 别人的东西」和「被别人 d 的东西」，这里 d 表示微分。

「d 别人的东西」是线性的 differential operators，记作 $\dfrac{\partial}{\partial x_1}$、$\dfrac{\partial}{\partial x_2}$ 等；它们很自然地组成一个 vector space $T_x M$。

「被别人 d 的东西」是一些线性的微分形式，记作 dx_1、dx_2 等；它们属於 $T_x M$ 的 dual space。

Manifolds

基本上「流形」的意思是「弯曲的空间」，它们局部地近似於 Euclidean 空间 $\mathbb{R}^n$，局部的座标可以分段用一组微分映射来描述，这些 maps 叫 charts。



(57)

Phase space

「相位空间」指的是力学系统里，i 个粒子的位置 x_i 和动量 p_i 合并而成的 $(\boldsymbol{x}, \boldsymbol{p})$ 空间。但 configuration space 指的是所有可容许的位置 $\boldsymbol{x}$ 的空间。

Vector fields, differential forms, Hamiltonian flow

根据 Hamilton 方程，再用微分的 chain rule 可以得到：

$$\frac{d}{dt} = \frac{\partial H}{\partial \boldsymbol{p}} \frac{\partial}{\partial \boldsymbol{x}} - \frac{\partial H}{\partial \boldsymbol{x}} \frac{\partial}{\partial \boldsymbol{p}} \tag{58}$$

它是一个微分算子，亦即是向量场；它有个特别的名字叫 Hamiltonian flow $\vec{H}$（很多书记作 X_H）：

$$\vec{H} := \frac{d}{dt} = \{\cdot, H\} \tag{59}$$

可以看出 $\vec{H}H = \{H, H\} = \frac{dH}{dt} = 0$ 就是能量守恒的形式。

Hamilton 系统的动态方程就是：

$$\dot{\boldsymbol{x}} = \vec{H} \tag{60}$$

所以在我们的智能系统中，RNN 可以看成是 $\vec{H}$。

$$\omega = \sum_i dx_i \wedge dp_i \tag{61}$$

$$\omega(\vec{H}, \cdot) = dH \tag{62}$$

我暂时不很明白它的意义。

我们说 Hamiltonian flow 保持 (preserve) 辛结构。假设 $\Gamma_t(\boldsymbol{x}_0) = \boldsymbol{x}(t)$ 描述 Hamiltonian flow 的轨迹；$\Gamma_0(\boldsymbol{x}) \equiv \boldsymbol{x}$。The **pullback** of ω along Γ is still ω:

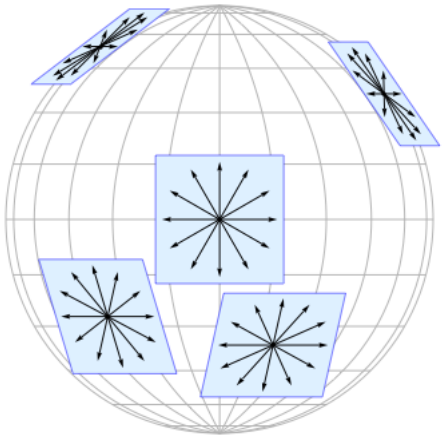
$$\Gamma_t^* \omega = \omega \tag{63}$$

$$\vec{H} H = 0 \quad \text{is equivalent to} \quad \Gamma_t^* \omega = \omega \tag{64}$$

Tangent and co-tangent bundle

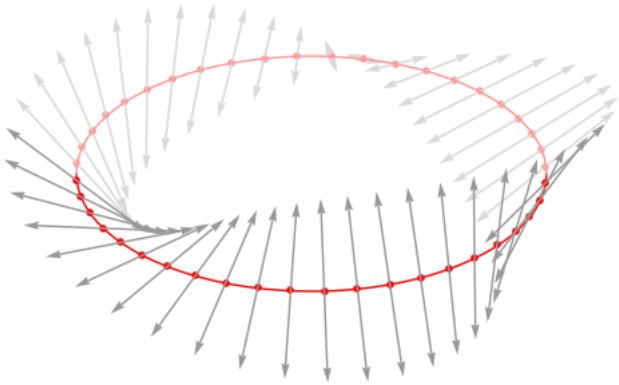
在流形上每点有一个 tangent space，所谓 tangent bundle 是指流形上每点 x 的 tangent space $T_x M$ 的总和：

$$TM := \bigcup_{x \in M} T_x M \tag{65}$$



Tangent bundle on a 2-sphere

(66)



Bundle with Möbius strip topology

(67)

粗略地，tangent bundle 可以看成是 $M \times T_x M$，而 M 和 $T_x M$ 的维数都是 n，所以 tangent bundle 的维数是 $2n$。在力学上，**cotangent bundle** 就是相位空间 $(\boldsymbol{x}.\boldsymbol{p})$ 的空间 (The phase space of a mechanical system is the cotangent bundle of its configuration space)。

Push forward, pull back

Energy conservation, area form

(Stephanie §4.4)

Symmetry of Hamiltonian system, Lie groups

Symplectic groups 是一些保存辛结构的变换 T 的群：

$$\omega(Tx, Ty) = \omega(x, y) \quad \forall x, y \in V \tag{68}$$

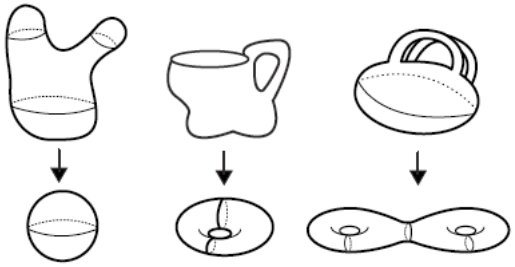
V 是向量空间。在 V 上的 symplectic 变换的全体记作 $Sp(V)$。

Momentum maps

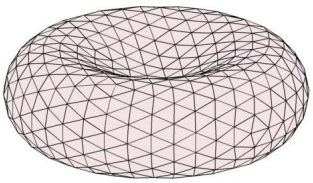
4.14 动态系统理论

Floer homology

Homology （同调论） 研究的是空间中「有没有穿洞」的楼拓结构。最简单的 **singular homology** 是将空间用三角形剖分 (triangulation)，然后透过著名的 Euler formula $V + F = E + 2$ 及其扩充，让我们可以计算 Euler characteristic χ，此即空间穿洞的个数。

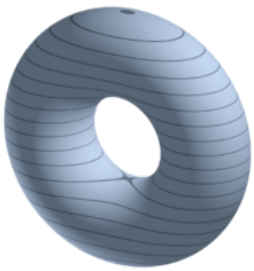


(69)



(70)

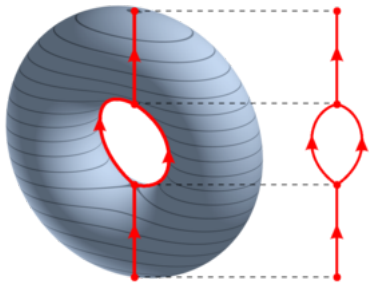
当这些三角形剖分趋於无穷小时，我们得到用微分形式 (differential forms) 描述的 homology，即 **de Rham homology**。



(71)

在流形上定义一个 potential function，例如简单的 height function，就可以做 Morse theory，这时每个点可以根据势能函数向下流 (gradient flow)，流到一些最低位置，它们是临界点 (critical points)。Morse decomposition 将空间用这些临界点分割（流到同一临界点的 flows 认作同一 equivalent class），得到的是

Morse homology。



(72)

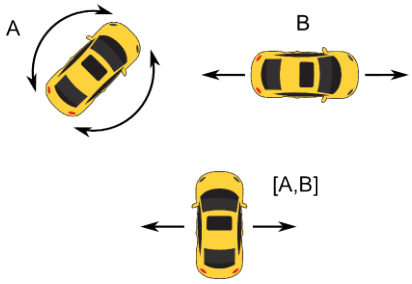
Floer homology 是 Morse homology 的**无限维空间**版本，比较难计算。

Conley theory

Entropy, ergodicity

Lie algebra 的另一应用

例如一架车可以有两个基本动作：(A) 绕中心旋转、或 (B) 前后行驶，它们的 Lie 括号 $[A, B]$，产生出的动作是左右方向行驶，这类似於「平行泊车」的时候，车子的移动方向：



(73)

（可控性与 “reachable” 概念。）

Stable, unstable, and center manifold

Smale horseshoe

5 最优控制的计算方法

直接法

间接法

Lyapunov 函数第一方法

Lyapunov 函数第二方法

6 关於逻辑....

由初始状态 x 过渡到 x'：

$$x \overset{\text{KB}}{\vdash} x' \tag{74}$$

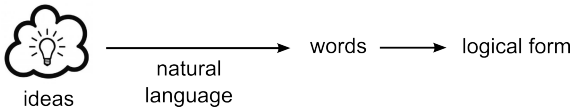
注意 x, x' 是命题的**集合**，$x, x' \subset \mathcal{P}(\mathcal{L})$，the power set of all logic formulas。

所谓 Q-value 其实是在 $X \times U$ 空间上的**能量分布**；$(x, u) \mapsto Q$。意思是说：在状态 x 执行动作 u，其价值是多少？（因为我们等同了价值和能量。）也可以说，Q 是 $X \times U$ 上的一个**测度** (measure)。

我们将逻辑推导中的**搜寻**纳入到 transition function 的功能之中。换句话说，transition function 的功能并不只是推导；它是一种类似 stochastic search 的「游荡」。

在逻辑 AI 这边，我们要计算的是分布在 $\overset{\text{KB}}{\vdash}$ 上的测度 Q；换句话说，对於不同的 KB，我们有不同的 $\vdash$。这似乎等於分布在所有不同的 KB 上的测度。

In LBAI the knowledge representation structure is built (*fixed*) from the bottom up:



(75)

but is it valid (or profitable) to assume that our mental representations are *isomorphic* to such logical structures? Or drastically different?

Humans are good at designing symbolic structures, but we don't know how to design *neural* representations which are more or less opaque to us. Perhaps we

could use a neural network acting recurrently on the state vector to **induce** an internal representation of mental space. "*Induced by what*," you ask? By the very structure of the neural network itself. In other words, forcing a neural network to *approximate* the ideal operator R^*.

From an abstract point of view, we require:

- R be an endomorphism: $X \to X$
- R has a learning algorithm: $R \overset{A}{\longmapsto} R^*$

R would contain all the knowledge of the KB, so we expect it to be "large" (eg. having a huge number of parameters). We also desire R to possess a **hierarchical** structure because hierarchies are computationally very efficient. A multi-layer perceptron (MLP) seems to be a good candidate, as it is just a bunch of numbers (weight matrices W) interleaved by non-linear activation functions:

$$R(\boldsymbol{x}) = \sigma(W_1\sigma(W_2...\sigma(W_L\boldsymbol{x}))) \tag{76}$$

where L is the number of layers. MLPs would be our starting point to explore more design options.

In 1991 Siegelmann and Sontag [5] proved that recurrent neural networks (RNNs) can emulate any Turing machine. In 1993 James Lo [2] proved that RNNs can universally approximate any non-linear dynamical system.

The idea of R as an operator acting on the state is inspired by the "consequence operator" in logic, usually denoted as Cn:

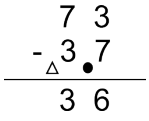
$$\text{Cn}(\Gamma) = \{ \text{ set of propositions that entails from } \Gamma \ \} \tag{77}$$

but the function of R can be broader than logical entailment. We could use R to perform the following functions which are central to LBAI:

- **deduction** – forward- and backward-chaining
- **abduction** – finding explanations
- **inductive learning**

Example 1: primary-school arithmetic

A recurrent neural network is a *much more powerful* learning machine than a feed-forward network, even if they look the same superficially.

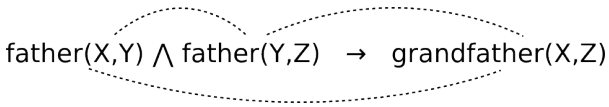


As an example, consider the way we perform 2-digit subtraction in primary school. This is done in two steps, and we put a dot on paper to mark "carry-over". The use of the paper is analogous to the "tape" in a Turing machine – the ability to use short-term memory allows us to perform much more complex mental tasks.

We did a simple experiment to train a neural network to perform primary-school subtraction. The operator is learned easily if we train the two steps *separately*. The challenge is to find an algorithm that can learn **multi-step** operations by itself.

Example 2: variable binding in predicate logic

The following formula in predicate logic defines the "grandfather" relation:



(78)

We did a simple experiment to train a neural network to perform primary-school subtraction. The operator is learned easily if we train the two steps *separately*. The challenge is to find an algorithm that can learn **multi-step** operations by itself.

In LBAI, logic possesses additional structure:

- **truth values** (eg. P(rain tomorrow) = 0.7)
- **propositional structure** (eg. conjunction: $A \wedge B$)
- **sub-propositional structure** (eg. predication: loves(john, mary))
- **subsumption structure** (eg. dog $\subseteq$ animal)

These structures can be "transplanted" to the vector space X via:

- **truth values:** an extra dimension conveying the "strength" of states
- **propositional structure:** eg. conjunction as vector addition,

$$A \wedge B = \boldsymbol{x}_A + \boldsymbol{x}_B + ... \tag{79}$$

but we have to avoid linear dependencies ("clashing") such as:

$$\boldsymbol{x}_3 = a_1\boldsymbol{x}_1 + a_2\boldsymbol{x}_2 \tag{80}$$

This would force the vector space dimension to become very high.

- **sub-propositional structure:** eg. tensor products as composition of concept atoms:

$$\text{loves(john, pete)} = \overrightarrow{john} \otimes \overrightarrow{love} \otimes \overrightarrow{pete} \tag{81}$$

- **subsumption structure:** eg. define the positive cone C such that

$$\text{animal} \supseteq \text{dog} \quad \Leftrightarrow \quad \overrightarrow{animal} - \overrightarrow{dog} \in C \tag{82}$$

But the more logical structure we add to X, the more it will resemble logic, and this whole exercise becomes pointless. Remember our original goal is to try something different from logic, by *relaxing* what defines a logical structure. So we would selectively add features to X.

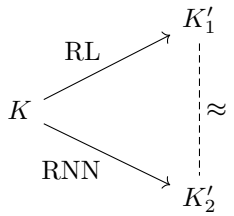
7 Unifying RL and RNN

From the viewpoint of reinforcement learning, we aim to learn the **policy** function:

$$\text{policy}: \quad \text{state} \xmapsto{\text{action}} \text{state'} \tag{83}$$

Where K can be regarded as the **mental state**, and thus an **action** in RL turns K into K'.

In our system, there are 2 pathways that act on K, via RNN and RL respectively:



(84)

In RL, the action a acts on K, whereas in RNN, R acts on K.

Note: RNN and RL are learning algorithms, and if they are both applied to the same problem, conflicts will necessarily arise, unless there is a way to combine them.

At state K, we estimate the Q-value $Q(K \overset{a}{\mapsto} K')$. The action that would be chosen at state K is $\arg\max_a Q(K \overset{a}{\mapsto} K')$. This could be used to train the RNN via $K \vdash_W ...^n K'$.

RL 在众多状态 K 之间游荡，学习 $Q(K \mapsto K')$。因为 RL 独有奖励讯息，我们必需用 RL 来教导 RNN 学习，反之不可。第一个问题是：RL 如何在 K 之间游荡？游荡是随机的，但也可以借助 RNN 的随机性、或在 RNN 自身的游荡中注入更多随机性、或者根本就是 RL 自己产生的随机性。接下来的问题是：RNN 如何用 Q 值来诱发学习？

RNN 的 "n-fold" 学习可以通过以下方式实现：

- stochastic forward-backward propagation
- genetic?
- 最有趣的是 Hebbian learning，因为它似乎特别适合这情况。

RNN 的本质是什么？它似乎是一个 recurrent hetero-associative memory。但其实它还需要将 input 作类似於 Word2vec 的 encoding。这个 encoding 将「相似」的思维状态 K 归到同类。利用空间中的相似度，RL 可以用一些连续函数来近似 Q 值（详细情况还有待分析）。

另一个问题是：虽然用函数的近似可以做到 generalization，但另一个方法是利用状态 K 中的空位作暂时储存。这两者似乎很不同。问题似乎在於：状态转换 $K \mapsto K'$ 是不是对应於逻辑中的一条 rule？答案似乎是 yes。这个共识是很重要的。如果用 decision tree，需要的是向量空间中的相似度。

现在的关键是「状态变量」。因为它可以做到符号逻辑中靠变量的 generalization，这是前所未有的。这种 generalization 似乎不需要相似度，因为它是符号的！会不会在向量空间中的状态变量能够做到之前逻辑变量做不到的动作？不管怎样，用 RNN 学习这些变量的动作似乎是很难的，因为这些动作似乎不是对误差的梯度下降。除非这些动作本身也近似於其他动作，但那是怎样的近似？学习 multi-step logic 其实和以前的 forward / backward chaining 没有分别！唯一分别是命题的 representation 改变了，它未必像符号的 concatenation。所以问题仍然是 "n-fold" 学习法。

而且注意：RL 的 generalization 根本上不同於 rules 空间中的 generalization。前者是思维空间 K 中的一般化，后者也可以是 K 空间的一般化，但也可以是依赖「状态变量」的一般化。

一般来说，RL 和 RNN 的行动和学习，是可以互相独立的。

还有 heterarchical 的分类法。想用 decision tree 或什么，达到不同网络的分工。在组织知识这方面，深度网络有没有用？可以想像，在视觉识别中，在网络的最上层有很多 objects，而它们都可以还原到底层的 features。网络有更多层，可以

识别的事物更抽象。但现在我们要的不是模式识别，而是 mapping。特别是抽象模式的 mapping。想要的是：大量的 rules，将不同的 K 映射到新的 K'。

还有一点要澄清的是：究竟每一个「思元素」在向量空间中是不是一点？如果有了这个「思元素 = 点」假设，则每次 iteration 应该会删除一个思元素，而用另一个（全新的）思元素取代之。这样，$K \mapsto K'$ mapping 就有了更确定的结构。这样的 setup 已经很接近 logic 系统，但其学习算法仍然很有 combinatorial 的 "feel"。（因为只有当两个 rules 串连之后，才能达到某个结论，而这个串连有没有中间的 continuous 状态?）这种串连通常是怎样找到的？

现在有一转机：如果「思元素 = 点」，则「状态变量」的形成似乎会很普遍，而我们可以集中研究如何学习 single-step rules。RL 的 rewards 可以指导学习，但这些「终极 rewards」对学习的细节没有指导作用。我们似乎可以用「时间延迟」来达到「状态变量」的效果，这个做法无形中增加了使用状态变量的机会。

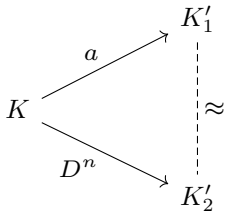
现在总结一下仍然有待回答的问题：

- RL 的 generalization 如何做？
- iterative thinking map 如何 learn？
-

Hebbian 的情况是：有某一 I/O pattern；我想 strengthen 这 pattern。

Assuming the learning is correct, K'_1 and K'_2 should be roughly the same — but this ignored the possibility that one path may take multiple steps to converge with the other path. [1]

Now I stipulate that R be more "refined", that is to say, applying D^n times may be equivalent to applying a once:



(85)

Using a different notation, a is the **restriction** or **section** of D^n at point K: $a = D^n|_K$.

Now the question is, do the RNN and RL paths have any *essential* difference?

[1] This situation has been encountered in term rewriting systems (TRS): If in a TRS any 2 different rewriting paths always converge to the same result, it is said to have the **Church-Rosser property**. For example the λ-calculus invented by Church has this property.

- Their internal **representations** are different:
  — RNN is a multi-layer neural network
  — RL's representation is $Q(\text{state}, \text{action})$, usually stored as a *look-up table*, although Q could be approximated by a neural network as well.
- RL learns through **rewards**, RNN learns from **errors**. Thus RL has broader applicability, because not all questions have "correct answers" that could be measured by errors. In RL we just need to praise Genifer whenever she displays good behavior.
- The internal cognitive state K exists because of RNN: it is simply the vector input and output of the RNN. Without this K, RL would be clueless as to what are its internal states. It can be said that the RNN provides a *machinery* for RL to control.

From the perspective of reinforcement learning, we could reward some results of multi-step inference:

$$x_0 \overset{a}{\longmapsto} x_{\vdash} \quad \updownarrow \bigstar \tag{86}$$

$\updownarrow \bigstar$ means "to give positive or negative rewards". We want to learn a which is the action to be taken at state K. The learning algorithm is based on the famous **Bellman optimality condition** (see next section).

Perhaps we should use RL to *guide* the learning in RNN, as RNN is more fine-grained....

To combine the 2 learning approaches, we could use the technique of **interleaving**: for each step apply RL once, apply RNN n times.

The learning in RNN may also involve **neurogenesis** (adding new neurons and connections), but I have not considered this aspect yet.

There are 4 learning modes:

- learning to listen/talk
- RL-based learning
- inductive learning

8 Misc points

- If sigmoid is replaced by polynomial, universal approximating property may be retained.

- Banach fixed point theorem does not apply because R in general need not be contractive. Question is whether R necessarily converges to fixed points and the answer is no.
- If reasoning operator R is continuous, the flow of the dynamical system is governed by an autonomous differential equation. Poincare-Bendixson only applies to dynamical systems on the plane, and is irrelevant to systems whose phase space has dimension ≥ 3, or to discrete dynamical systems.
- Time can be discrete or continuous.
- Goal is to find minimizer of error (ie, to approximate a function given some input-output data points). The (finite) set of local minima can be solved via setting $\frac{\partial R}{\partial W} = 0$. The number of local minima can be calculated as: ? McClelland paper.
- If operator is discontinuous, what advantages can be gained?

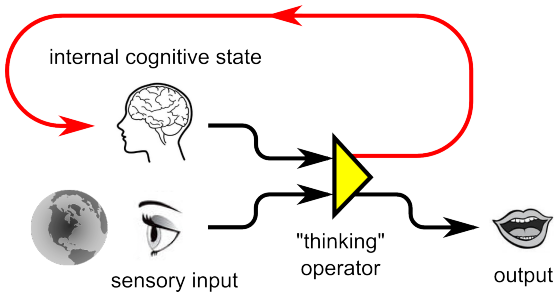
What I want to do now is to determine if R implemented as a deep network is sufficient to model human-level reasoning.

One principle seems to be that logical conclusions must not proliferate indefinitely. But we are not sure what kind of structural constraints this would impose on the vector space. Or whether we should impose such constraints manually.

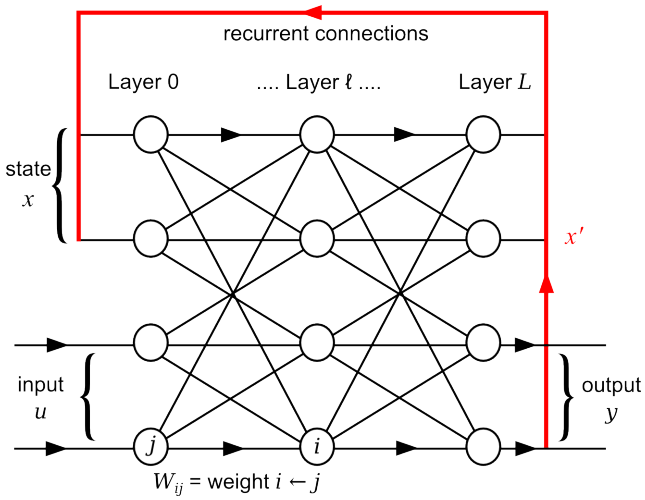
What other properties are desired for the implementation of R?

9 Architecture

First, cartoon version:

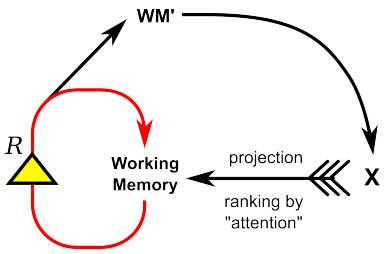


(87)



(88)

TO-DO: The state space X may be too large and we may need an **attention mechanism** to select some parts of X for processing by R. This is the notion of **working memory** in cognitive science.



(89)

10 Deep Recurrent Learning

The learning algorithm for R is central to our system. R learns to recognize input-output pairs $(\boldsymbol{x}_0, \boldsymbol{x}^*)$. What makes it special is that R is allowed to iterate a *flexible* number of times before outputting an answer. In feed-forward learning we simply learn single-pass recognition, whereas in common recurrent learning we train against a *fixed* time sequence. Here, the time delay between input and output is allowed to stretch arbitrarily.

Suppose the recurrent network R iterates n times:

$$\boldsymbol{x}_{t+1} = \overbrace{R \circ R \circ ...}^{n}(\boldsymbol{x}) \tag{90}$$

As $n \to \infty$, we get the continuous-time version (a differential equation):

$$\frac{d\boldsymbol{x}(t)}{dt} = \mathfrak{R}(\boldsymbol{x}(t)) \tag{91}$$

We could run the network R for a long enough time T such that it is highly likely to reach an equilibrium point. Then:

$$\boldsymbol{x}_T = \int_0^T \mathfrak{R}(\boldsymbol{x}(t))dt \tag{92}$$

and the error:

$$\boxed{\text{误差}} = \boldsymbol{x}^* - \boldsymbol{x}_T \tag{93}$$

where $\boldsymbol{x}^*$ is the target value which is independent of time.

$$\begin{aligned}\frac{\partial \mathscr{E}}{\partial \boldsymbol{W}} &= -\frac{\partial}{\partial \boldsymbol{W}} \int_0^T \mathfrak{R}(\boldsymbol{x}(t))dt \\ &= -\frac{\partial}{\partial \boldsymbol{W}} \int_0^T \textcircled{\scriptsize$\int$}(W_1 \textcircled{\scriptsize$\int$}(W_2 ... \textcircled{\scriptsize$\int$}(W_L \boldsymbol{x}(t)))dt\end{aligned} \tag{94}$$

When there are many layers or if the recurrence is too long, back-prop learning becomes ineffective due to the **vanishing gradient** problem. One solution is to use the **rectifier** activation function:

$$\textcircled{\scriptsize$\diagup$}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{95}$$

Since its derivative is piecewise constant, it does not suffer from the vanishing gradient problem.

Acknowledgements

In a forum discussion with Ben Goertzel dated 25 June 2014 on the AGI mailing-list: (artificial-general-intelligence @googlegroups.com), YKY asked: Why bother with neural networks, which typically require many neurons to encode data, when logic-based AI can represent a proposition with just a few symbols? Ben's insight is that neural networks are capable of learning its own representations, and their learning algorithms are relatively fast. We have been working on "neo-classical" logic-based AI for a long time, and begin to realize that inductive learning in logic (based on combinatorial search in a symbolic space) is perhaps *the bottleneck* in the entire logic-based paradigm. So we try to look for alternatives that might enable learning to be faster, though we would still emphasize that logic-based AI remains a viable approach to AGI.

References

1. Itamar Arel. *Deep reinforcement learning as Foundations for Artificial Intelligence*, chapter 6, pages 89–102. Atlantis Press, 2012.
2. Lo. Dynamical system identification by recurrent multilayer perceptrons. *Proceedings of the 1993 World Congress on Neural Networks*, 1993.
3. Mikolov, Sutskever, Chen, Corrado, and Dean. Efficient estimation of word representations in vector space. *Proceedings of workshop at ICLR*, 2013.
4. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
5. Siegelmann and Sontag. Turing computability with neural nets. *Applied Mathematics Letters, vol 4, p77-80*, 1991.